INF2050 – Outils et pratiques de développement logiciel

Construction automatisée

Jacques Berger

Objectifs

Introduire les outils de construction de logiciels

Prérequis

Gestion des sources

Construction de livrables

Le livrable le plus important : le logiciel

Plusieurs étapes peuvent être nécessaires afin de construire ce livrable et le préparer pour son déploiement

Construction de livrables

L'environnement de développement n'est pas toujours exactement le même que l'environnement de production

Par exemple : on ne déploie pas les sources, uniquement les exécutables

Étapes possibles

Génération de code
Vérification du style
Compilation
Édition des liens (link)
Génération du bytecode (Java)
Compilation et génération de la documentation
Création des librairies (jar, dll)
Création de répertoires, copie de fichiers
Modifier les permissions de fichiers et répertoires
Exécution des tests

Outils de build

Toutes ces manipulations peuvent être faites manuellement

Ce travail manuel est répétitif et sujet aux erreurs et oublis

Certains outils nous permettent d'automatiser toutes les manipulations qui touche la production du logiciel

Outils de build

Quelques outils connus :

make
Ant
Maven
Buildr
Gradle

Fonctionnement

On part d'un répertoire vide

On construit le logiciel entièrement à partir des sources

On produit les livrables nécessaires

Build habituel

On commence par un checkout des sources

Construction du projet
    Compilation, link, etc

Déploiement
    Permissions, documentation, readme, librairies

Tests
    Unitaires, fonctionnels, régression

Build

Certains outils permettent d'aviser une personne par courriel si le build a échoué

Afin de faciliter l'utilisation de l'outil de build et faciliter son intégration à d'autres outils, il est souhaitable de pouvoir l'utiliser à la ligne de commande

Build

Certaines tâches sont très communes (ex. compilation, tests unitaires) mais d'autres peuvent être spécifiques à une entreprise

Un outil permettant de créer des tâches personnalisées et de les ajouter au build apporte une flexibilité considérable

Gestionnaire de sources

Un outil de build combiné à un gestionnaire de sources peut construire n'importe quelle version actuelle ou antérieure (ex. pour une date donnée)

Portabilité

Avant les outils de build, on automatisait le build avec des scripts dépendants du système d'exploitation

Les outils d'aujourd'hui sont plus portables et nous permettent de construire le logiciel sur à peu près n'importe quel système

Tests

Le build complet permet de solidement tester l'intégration d'une modification

On peut construire l'application plusieurs fois par jour

Un build devrait se compléter dans un temps raisonnable (certains disent en moins de 10 min)

Tests

Intégrer l'exécution des tests unitaires au build est une façon efficace de vérifier si le build est bon

Ceci implique la présence d'une bonne couverture de tests unitaires

Avantages

Permet une construction plus fréquente et plus rapide

Meilleure qualité des livrables
- Moins d'erreurs de construction
- Moins d'oublis

Avantages

Automatiser des tâches redondantes
- Moins de coûts
- Moins de ressources

Production de rapports d'erreurs et exécution de tests

make

Outil Unix pour scripter des commandes de construction de logiciels

Les commandes sont des commandes Shell, donc spécifiques à la plateforme qui l'exécute

make

Le script s'appelle un makefile

On divise les différents morceaux à construire en cibles (targets)

L'outil exécute les cibles uniquement si nécessaire

make

La cible spécifie ses dépendances (d'autres cibles qui doivent être exécutées avant celle-ci) et les commandes à exécuter

La syntaxe :

```
nom_de_cible: dependance
    Commande#1
    Commande#2
```

make

Il est également possible de spécifier des constantes pour éviter la répétition dans les commandes

Voir un exemple

make

Très rapide mais limité à la plateforme qui l'exécute

N'est pas utile pour la publication du logiciel (open-source) mais peut être pratique en développement

Très utilisé avec les vieux projets C et C++

Ant

Projet Apache

Un make plus moderne

Portable
- Syntaxe XML
- Exécute du code Java

Ant

Fonctionne à peu près de la même façon que make au niveau des cibles et dépendances

Pour la portabilité, on évite de mettre des commandes Shell dans un script Ant

On utilise des tasks au lieu des commandes

Ant

La plupart des manipulations courantes sont disponibles au travers d'une task Ant

Pour les choses plus spécifiques, on peut créer une task en Java et l'utiliser avec Ant

Ant

Le script doit être placé à la racine du projet et le fichier doit s'appeler build.xml

Ainsi, la plupart des IDE vont le supporter directement

Ant

Ant a été très populaire dans le passé, principalement avec les logiciels libres

NetBeans utilise Ant à travers son UI

Sa syntaxe est complexe et verbeuse, ce qui a entraîné une certaine perte de popularité

Ça demeure un bon outil, très utile

Maven

Projet Apache

Version courante : Maven 3.6

Construction plus évoluée que Ant

Maven

Convention over configuration

Si on suit les règles préétablies par Maven, il n'y a presque pas de configuration à faire pour le faire fonctionner

Maven

Permet :

- La construction
- Le déploiement
- L'exécution des tests
- La gestion des dépendances

Maven

La configuration de Maven se retrouve dans le document POM

Project Object Model

Il s'agit d'un document XML qui contient tout ce qui est spécifique au projet

Maven

Maven normalise la structure du projet

- Répertoires
- Packages
- Tests

Maven

La gestion des dépendances peut poser problème dans un projet en Java

- Librairies JAR

Difficile de savoir quelle version du JAR on doit utiliser, et on ne veut pas nécessairement mettre le JAR dans le gestionnaire de sources

Maven propose une solution à ce problème

Maven

On peut spécifier les dépendances du projet dans le document POM :

```
<dependencies>
    <dependency>
        <groupId>isorelax</groupId>
        <artifactId>isorelax</artifactId>
        <version>20030108</version>
    </dependency>
    <dependency>
        <groupId>com.sun.xml.bind</groupId>
        <artifactId>jaxb1-impl</artifactId>
        <version>2.2.5.1</version>
    </dependency>
</dependencies>
```

Maven

Les librairies sont gérées par un repository Maven

Ce repository peut être publié dans un réseau, sur une machine locale ou sur Internet

Maven

Généralement, le fournisseur de la librairie fournit l'élément dependancy à ajouter au POM sur son site web

Sinon, ce site peut être très utile
http://mvnrepository.com/

Maven

Maven ajoutera automatiquement au projet toutes les librairies dont dépendent celles du POM

On appelle celles-ci des dépendances transitives

Maven

Exemple :

En ajoutant une dépendance sur la librairie :
    json-lib-2.4-jdk15

Maven ajoute automatiquement les librairies :
    commons-beanutils-1.8.0
    commons-collections-3.2.1
    commons-lang-2.5
    commons-logging-1.1.1
    ezmorph-1.0.6

Maven

Maven offre beaucoup plus de fonctionnalités mais sa gestion des dépendances en fait un outil intéressant pour les projets Java

Il s'utilise à la ligne de commande ou s'intègre à votre IDE

Maven

Attention à l'élément classifier dans le POM, ceci pourrait vous sauver des heures de recherche

```
<dependency>
    <groupId>net.sf.json-lib</groupId>
    <artifactId>json-lib</artifactId>
    <version>2.4</version>
    <classifier>jdk15</classifier>
</dependency>
```

Nouvelle génération

Scripts basés sur des langages dynamiques

Gradle (Groovy)

Buildr (Ruby)

Plus loin...

GNU Make
    http://www.gnu.org/software/make/

Apache Ant
    http://ant.apache.org/

Apache Maven
    http://maven.apache.org/

Dépôt Maven
    http://mvnrepository.com/